Turma 2DL Grupo 03

Professor

Bruno Silva (BAS)

Unidade Curricular

Redes de Computadores RCOMP

Relatório RCOMP

WebWalls Project

André Santos 1150968
David Santiago 1161109
Hugo Fernandes 1161155
João Santiago 1160696

Parte Inicial

Este relatório tem como objetivo apresentar o segundo projeto de RCOMP, nomeadamente a realização de uma aplicação na qual é possível a criação de mensagens e de walls através de linguagens de programação como o Java, JavaScript e HTML para a criação do website, proposto pelos professores desta cadeira. Assim, propomos esta versão com o intuito de chegarmos à melhor solução possível.

Índice

Índice de Figuras

1. Introdução

Este projeto está a ser realizado no âmbito da unidade curricular de Redes e Computadores. Numa primeira aplicação, a mais complexa, vamos ter um HTTP server (servidor) e este vai manipular as walls possibilitando a REST web services e também uma interface de mensagens UDP (cliente). Numa segunda aplicação teremos um simples cliente UDP que permite a simples escrita de mensagens numa Wall utilizando a interface UDP, isto tudo utilizando as bases que foram lecionadas nas aulas de RCOMP. Assim, apresentaremos uma resolução em Java na qual é possível verificar este processo.

2. Modo de Funcionamento

Para a implementação deste projeto utilizamos alguns métodos e classes já fornecidos pelos professores no moodle e, com isto, desenvolvemos a partir daí a nossa aplicação, desenvolvendo os nossos próprios métodos nas variadas linguagens utilizadas, tendo sido a página web também criada por nós. O HTTP server (servidor) foi onde implementamos os métodos de GET, POST E DELETE e onde possibilitamos os variados tipos de conteúdo: text/plain, text/html, image/gif, image/png, application/javascripte e application/xml.

3. Web main page loading (user's fronted)

A página web principal do HTTP server (GET /) que é a index.html foi desenhada em HTML e todos os casos de uso nela implementados foram desenvolvidos em JavaScript e agem como um consumer (AJAX).

3.1 Casos de uso

- Alterar a Wall existente.
- Adicionar uma mensagem à Wall existente.
- Remover uma mensagem da Wall existente.
- Remover a Wall existente.

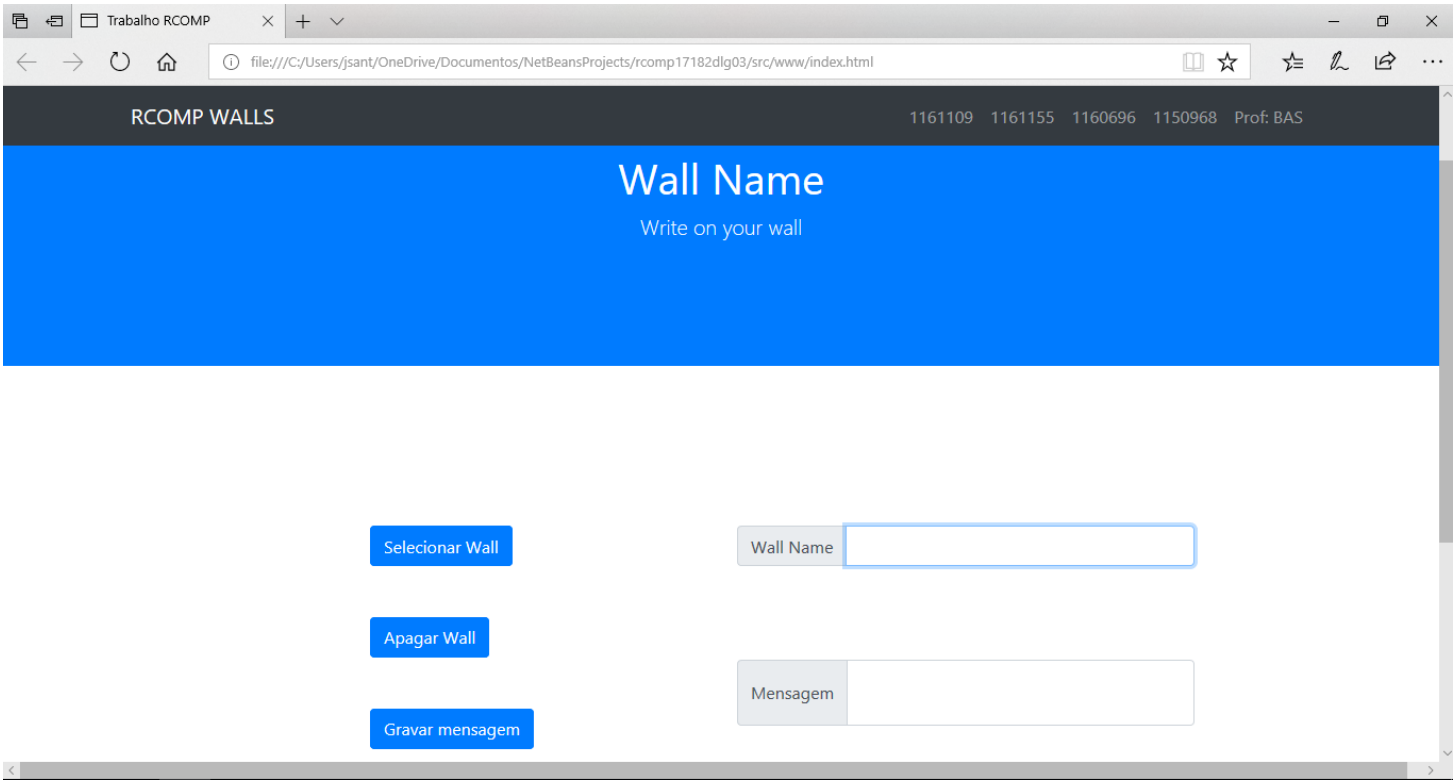

Figura 1- Página Web Principal

4. REST web services

Cada Wall é identificada por um nome e numa Wall devem ser produzidas as mensagens e estas têm que ter um número de sequência implementado pela aplicação do server.

Em cada Wall, a primeira mensagem nela escrita deve ter o número zero e as próximas mensagens vão ter sempre o número da anterior incrementado por um. Se, eventualmente o server é parado todas as mensagens e walls são perdidas automaticamente pois não se trata de um server persistido.

Assim, os web services REST API foram implementados seguindo a seguinte estrutura:

/walls/{WALL'S-NAME}/{MESSAGE-NUMBER}

Onde {WALL'S-NAME} é o nome da Wall e {MESSAGE-NUMBER} é o número da mensagem.

4.1 Casos de uso REST

- Adicionar uma mensagem: POST /walls/{WALL'S-NAME}
- Apagar uma mensagem: DELETE /walls/{WALL'S-NAME}/{MESSAGE-NUMBER}
- Apagar uma Wall: DELETE /walls/{WALL'S-NAME}
- Listar as mensagens: GET /walls/{WALL'S-NAME}

4.2 Algumas informações importantes

1. GET /walls não é permitido pelo server pois os nomes das walls não são públicos.
2. O método PUT não é implementado pois as mensagens não sofrem alterações depois de publicadas.
3. Ao apagar uma Wall consequentemente são apagadas todas as mensagens nela existentes.

4. Uma Wall é sempre criada automaticamente independentemente de estar criada ou não, pois ao ser feita um POST de uma mensagem vai ser criada uma Wall para esta.

5. Low Level UDP messages interface

Este serviço, o UDP messages interface, está disponível através de uma porta que nos foi atribuída pelo professor, porta essa que é 32113.

O único caso de uso disponível pelo UDP é o adicionar uma mensagem a uma Wall, que é igual ao POST no web services REST interface. Para isto, todos os requests e respostas são nomeadas segundo uma estrutura.

O cliente UDP está destinado a enviar requests ao ip broadcast address local.

6. Explicação do código

POST

Classe HttpServerWalls:

```
//por a receber nome da wall se nao existir invoca o metodo de criar a wall
public static void addMsg(String uri) {
    System.out.println("URI:" + uri);
    synchronized (walls) {
        String s[] = uri.split("/");
        String nameW = s[2];
        String msg = s[3];
        System.out.println("Mensagem a adicionar: " + msg);
        boolean condition = false;

        for (int i = 0; i < walls.size(); i++) {
            if (walls.get(i).getName().equals(nameW)) {
                walls.get(i).getMsgs().add(msg);
                System.out.println("Adicionou mensagem a wall existente");
                condition = true;
            }
        }

        if (condition == false) {
            System.out.println("criou nova Wall");
            System.out.println(nameW);
            Wall w = new Wall(nameW);
            w.getMsgs().add(msg);
            walls.add(w);
        }

        walls.notifyAll(); // notify all threads waiting on MSG_LIST's monitor
    }
}
```

Figura 2- Adicionar uma mensagem POST

Classe HttpWallsRequest:

```
} else { // NOT GET
    if (request.getMethod().equals("POST") && request.getURI().startsWith("/walls/")) {
        System.out.println("entrou no post");
        HttpServerWalls.addMsg(request.getURI());
        response.setResponseStatus("200 Ok");
```

Figura 3- POST message request

DELETE

Classe HttpServerWalls:

```
//metodo delete a uma wall
public static void deleteWall(String uri) {
    synchronized (walls) {
        String s[] = uri.split("/");
        String w = s[2];

        for (int i = 0; i < walls.size(); i++) {
              try {
                  walls.wait();
              } // wait for a notification on MSG_LIST's monitor
              // while waiting MSG_LIST's intr lock is released
              catch (InterruptedException ex) {
                  System.out.println("Thread error: interrupted");
              }
            if (walls.get(i).getName().equalsIgnoreCase(w)) {
                walls.remove(i);
            }
        }
    }
}
```

Figura 4- Apagar uma Wall DELETE

Classe HttpWallsRequest:

```
else if (request.getMethod().equals("DELETE") && request.getURI().startsWith("/walls/") && (request.getURI().split("/").length == 3)) {
    System.out.println("Entrou no delete wall");
    HttpServerWalls.deleteWall(request.getURI());
    response.setResponseStatus("200 Ok");
```

Figura 5- DELETE wall request

DELETE

Classe HttpServerWalls:

```
//metodo delete a uma msg
public static void deleteMessage(String uri) {
    String s[] = uri.split("/");
    String w = s[2];
    int index = Integer.parseInt(s[3]);

    for (int i = 0; i < walls.size(); i++) {
          try {
              walls.wait();
          } // wait for a notification on MSG_LIST's monitor
          // while waiting MSG_LIST's intr lock is released
          catch (InterruptedException ex) {
              System.out.println("Thread error: interrupted");
          }
        if (walls.get(i).getName().equalsIgnoreCase(w)) {
            ArrayList<String> ms = walls.get(i).getMsgs();
            if (ms.get(index) == null) {
                System.out.println("Mensagem nao existe.");
            }
            ms.set(index, "");
        }
    }

}
```

Figura 6- Apagar uma mensagem DELETE

Classe HttpWallsRequest:

```
else if (request.getMethod().equals("DELETE") && request.getURI().startsWith("/walls/") && (request.getURI().split("/").length == 4)) {
    System.out.println("Entrou no delete wall");
    HttpServerWalls.deleteMessage(request.getURI());
    response.setResponseStatus("200 Ok");
```

Figura 7- DELETE message request

7. Conclusão

Em suma, o projeto que realizamos aplica todos os requisitos necessários e apontados no enunciado. Assim, utilizando os conhecimentos aprendidos nas aulas, conseguimos desenvolver uma aplicação bem estruturada e organizada que funciona da forma pedida.